One Solution to Rule Them All: ATTEST as Unified Testing Solution for Programming Courses

Meinhard Kissich
meinhard.kissich@tugraz.at
Graz University of Technology
Graz, Austria

Kristóf Kanics
kristof.kanics@tugraz.at
Graz University of Technology
Graz, Austria

Klaus Weinbauer
klaus.weinbauer@student.tugraz.at
Graz University of Technology
Graz, Austria

Tobias Scheipel
tobias.scheipel@tugraz.at
Graz University of Technology
Graz, Austria

Marcel Baunach
baunach@tugraz.at
Graz University of Technology
Graz, Austria

ABSTRACT

Assessing student submissions and providing accurate and timely feedback in educational courses that involve writing code has shown to be demanding. Thus, we recently designed an automated test system that builds submitted code, runs it on the actual target hardware, and generates a report that may include the results of various measurement devices. After a successful test run in the Real-Time Operating Systems course, we aim to adopt our ATTEST test system in further courses spanning the whole system stack. A unified testing solution benefits students by lowering extraneous cognitive load and supervisors by reducing maintenance effort. This work-in-progress paper states why test cases remain an essential assessment technique, summarizes approaches to virtualize laboratory courses, and elaborates on the course-specific testing requirements and expected outcomes.

CCS CONCEPTS

• **Computer systems organization** → *Real-time operating systems*; *Embedded systems*; • **Applied computing** → **Education**.

KEYWORDS

student assessment, test system, programming education, digital design

1 INTRODUCTION

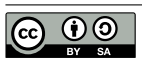
As the number of embedded devices is expected to grow significantly [14], we envision a high demand for well-educated embedded software (and hardware) engineers. Thus, our lecture complexes on embedded system design address topics across the entire system stack, from processor architectures over operating system design to writing efficient and reliable embedded application software.

Mentorship. Experience showed that timely and accurate feedback contributes to learning success and motivation throughout the course. While we support the courses with a comprehensive script and Q&A hours, an adequate relationship between students and tutors to respond to questions about implementation details timely is often challenging.

Test system. To support students further in their incremental improvements and finding bugs, we provide a test system in the Real-Time Operating System (RTOS) course that builds students' code and executes it on the actual target hardware together with carefully crafted test cases. Based on the experience gained from our first test system, we worked out the education-specific requirements and redesigned the test system from scratch in the winter semester of 2022/2023. The newly designed ATTEST [9] test system was later made open source under a permissive license and builds a versatile core module for testing and assessing submissions in various educational courses.

Adoption in further courses. As we had a stable operation and received remarkable feedback on the first semester in use, we now aim to adopt the test system for further courses we offer. Modularity and extensibility have been the main design criteria [9] to (i) provide a uniform interface for several courses, thus reducing extraneous cognitive load [7], (ii) minimize the maintenance effort, (iii) provide testing hardware-dependent code without local target hardware, and (iv) provide a setup that delivers reliable assessment independent of the local toolchain and hardware.

Contributions and paper outlook. This work-in-progress paper outlines the strategy to apply ATTEST to further courses and make it a unified testing solution. It elaborates on the required modification in the setup to cover the varying testing requirements, from testing digital designs over evaluating operating systems and basic software code to



 DOI: https://doi.org/10.18420/fgbs2023h-01. FGBS '23, September 28–29, 2023, Bamberg, Germany.

testing application software. The remainder of the paper is organized as follows. Section 2 discusses related work and alternative approaches to on-site laboratories and mentoring. Section 3 covers course-specific details and is split into three subsections. We first recap the experience gained in the Real-Time Operating System course (Section 3.1). We then discuss how to efficiently use ATTEST for a digital design course that requires hardware simulation, synthesis and software tests (Section 3.2) and for a course on Real-Time Bus Systems that requires building electronics for a CAN transceiver (Section 3.3). Finally, Section 4 concludes the paper by summarizing the expected results.

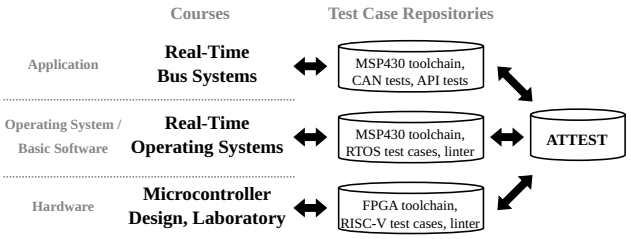


Figure 1: Courses that are going to use ATTEST. Embedded system stack layers (left), courses and corresponding test case repositories (middle), and the inclusion of the unified test system (right).

2 RELATED WORK

Artificial intelligence as a virtual mentor. The rapidly advancing achievements in artificial intelligence bring new opportunities in education. Blocklove et al. [3] use Large-scale Language Models (LLMs) to generate digital designs and the respective testbench code. While simple designs have been created satisfactorily with some guidance, the tools repeatedly had issues crafting correct testbench code. Tian et al. [13] performed an empirical study using Artificial Intelligence (AI) as a programming assistant. As of now, AI has trouble generalizing well to unseen tasks, and developers should not rely on synthesized code for non-trivial problems. Also, the response must be reviewed manually when AI is used to repair code snippets. In education, we aim to convey a precise, in-depth, and unambiguous understanding of the covered topics. Gulwani et al. [6] present an interesting approach to providing students with more guided feedback by repairing the submitted results. Still, test cases remain unavoidable to assess submissions and obtain sufficient reference implementations. Thus, testing fine-grained and carefully designed test cases remains necessary and reflects how software engineers test their implementation.

Remote laboratories. According to our experience in laboratory courses, students actively use the possibility of working from home at flexible hours instead of completing assignments on-site. Del Villar et al. [15] come to the same conclusion after setting up a learning environment for the Controller Area Network (CAN) communication protocol using LabsLand's [2] remote laboratories. Fotopoulos et al. [4] propose a remote laboratory as well for digital electronic design to increase the educational impact. In contrast to these approaches, our ATTEST test system is fully open-source and allows lecturers to evaluate students' work progress based on resulting test reports.

RISC-V related test suites. While submissions in programming courses usually need to satisfy the exercise specification only, digital designs of RISC-V processors must also match an external Instruction Set Architecture (ISA) specification. The community openly provides such tests in the form of test suits. RISCOF [12] is a framework that uses a set of architectural assembly test cases [5] to check compliance with the RISC-V specification. Similarly, the official RISC-V tests [8] provide unit tests for various cores. While the presented frameworks are not directly designed to be used for teaching, they form a basis for supporting a RISC-V digital design course by ATTEST.

3 COURSES AND TEST SYSTEM SETUPS

This section outlines how to use ATTEST as a unified test system and elaborates on the course-specific requirements. While RTOS was our first course that used an automated test system, new demands range from testing digital designs to network transceivers accessed from the application layer through the entire software stack. Figure 1 classifies our courses according to the system layer they belong to. Moreover, it shows the corresponding test case repositories and emphasizes that the unified ATTEST core is used in all cases.

3.1 Real-Time Operating Systems

In the RTOS course, students design and implement a lightweight version of our embedded research operating system, *SmartOS* [11], in C and assembly. The test setup is detailed in [9] and summarized in Figure 2. The test system receives the student's submission via a Git repository. After building, ATTEST executes the test cases using an appropriate test unit, i.e., a stand-alone MSP430 board or an MSP430 board connected to an oscilloscope – depending on the test case requirements. Three types of test cases are supported, as shown in Table 1. *Static tests* perform operations on the submission that do not require a target hardware. Examples are tracking the size of the built binary or checking conformance to a coding style (linting). *Message tests* execute the test case on the target hardware and compare received

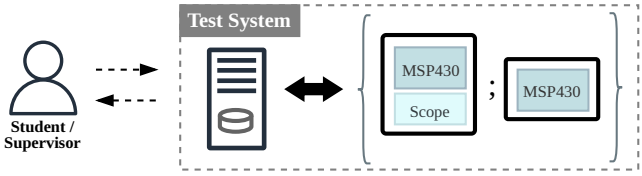


Figure 2: RTOS course test system setup. The host computer uses one of the available test units capable of executing the corresponding test case.

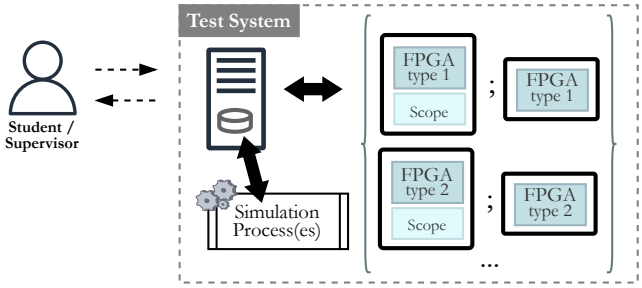


Figure 3: MDLab test system setup. The host computer uses one of the available test units capable of executing the corresponding test case. In addition, the host features a simulation process to run simulation tests (*Sim tests*).

messages against a reference, yielding a pass or fail. *Message tests* require only an MSP430 board, while *probe tests* additionally require an oscilloscope to monitor physical signals. The *probe tests* check the OS timing and evaluate the performance of various benchmarks.

Feedback and learnings. An early evaluation of the course feedback revealed significant improvements in the test system's performance and a high usage rate. Due to parallelization, the throughput could be increased to lower the average time to receive a test report. Eight test units are currently in operation, three of which are equipped with a measuring device. Together, the 48 participants have performed over 2500 test runs, each consisting of 13 to 57 individual test cases (depending on the progress of the course).

In addition to the direct benefits to students, supervisors can also take advantage to optimize their course. For example, correlating test cases can be identified, i.e., pairs of test cases that either both pass or fail. Thus, the test coverage can be optimized without lowering the experienced resolution, increasing test time, or raising the number of programming cycles. Another facet addresses partitioning the overall OS implementation into individual and manageable exercises. Having an in-depth insight into the number of (re-)submissions and test runs can reveal exercises that require more effort to be robustly implemented. Thus, exercise and deadline planning can be optimized.

3.2 Microcontroller Design, Laboratory

In the Microcontroller Design, Laboratory (MDLab), students design and implement a RISC-V processor core with four pipeline stages according to the RISC-V ISA [16] using SystemVerilog [1]. The processor core is modularized to maximize the learning outcomes. The final core is then embedded into a Microcontroller Unit (MCU) that connects to several on-chip peripherals through a Wishbone interconnect [10]. Due to its modularity, the MCU can be fully simulated and even synthesized for an Field-Programmable Gate Array (FPGA) at every implementation step. The types of test cases to check the submissions for an implementation step are shown in Table 2. The related test setup is depicted in Figure 3. While some test cases only use simulation processes, hardware tests can be carried out on several different FPGA types, stand-alone or together with an oscilloscope connected to the FPGA's pins. Similarly to RTOS, the test unit is selected according to the test case requirements. *Static tests* perform operations on the submission that do not invoke any test unit. Up to now, coding style tests (linting) have been envisioned. All further tests are based on existing RISC-V ISA test programs that are executed in different ways: *Sim tests* invoke hardware simulation processes and compare the simulation results against a reference. In addition, the submitted designs are sanity-checked by using smoke tests. In contrast to *static tests*, the simulation processes may need extensive computing power and could be executed on external servers. *Message tests* and *probe tests* execute the test case on one or more FPGA types and compare the received messages against a reference. RISC-V ISA test programs and smoke tests are planned for both test types. As far as hardware is concerned, *message tests* and *probe tests* for the MDLab work the same way as in the RTOS course, but with different FPGA boards instead of the MSP430 board.

Table 1: Used test case classes in the RTOS course and considered improvements (italics).

Static Tests	Message Tests	Probe Tests
Binary size	Correctness tests	Timing checks
Coding style		Performance benchmarks

Table 2: Proposed test case classes in the MDLab.

Static Tests	Sim Tests	Message Tests	Probe Tests
Coding style	RISC-V ISA test programs		
	Correctness tests		Timing analysis
		Smoke tests	

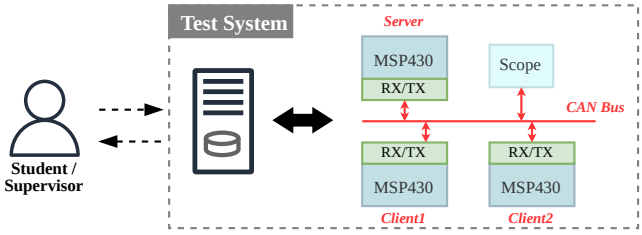


Figure 4: RTBS course test system setup. The host computer executes the test cases on one of the MSP430 boards or initiates communication among all three nodes and performs measurements.

3.3 Real-Time Bus Systems

In the Real-Time Bus Systems (RTBS) course, students build a communication network using the CAN protocol. While the first exercise deals with implementing simple supporting functions (e.g., bit manipulations), subsequent exercises increase in complexity, building on top of the previously implemented functionality. After finishing the communication stack, multiple nodes are connected to a common CAN bus to establish inter-node communication. Due to this gradual progression, students have to make sure they can rely on the previously implemented steps to save time on debugging later on. Similarly to RTOS, the target hardware used is an MSP430 board. In addition, a CAN transceiver extension module is used to interface the CAN bus. Thus, the setup has only minor deviations from the setup used in the RTOS course, and the adoption is expected to be low-effort.

RTBS will utilize *message* and *probe tests* for testing the correctness of the implementation, as well as *static tests* for checking the coding style, as shown in Table 3. While the correctness test of the communication stack invokes *message tests* on one test unit, testing the entire network requires three test units and an oscilloscope connected to each one. Each of the test units is equipped with a CAN transceiver extension module. As Figure 4 depicts, three bus participants are configured in the network connected to the test system: Server, Client1, and Client2. Each node will be able to act as any bus participant. Two nodes will run the reference implementation, while the third node will execute the student's code. ATTEST's *Scheduler* can decide which node takes over which role to minimize the number of programming cycles. Further *message* and *probe tests* will be performed to check the correctness (including collision detection) and give feedback about the timing behavior.

Table 3: Proposed test case classes in the RTBS course.

Static Tests	Message Tests	Probe Tests
Coding style	Correctness tests	
		Timing analysis

4 EXPECTED OUTCOMES AND CONCLUSION

Our ATTEST test system successfully operated in the past semester in the RTOS course and was well received by students and tutors. Thus, we aim to adopt it to further courses that demand writing and evaluating code for embedded hardware. In this paper, we emphasized the importance of test cases and presented our strategy to make ATTEST a unified testing solution. We elaborated on course-specific testing requirements and hardware setups. Although the courses cover the whole system stack, adoption is considered to be possible with reasonable effort. We expect to improve the overall learning experience, support the tutors in assessing the assignments, and motivate students.

REFERENCES

[1] 2018. IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language. *IEEE Std 1800-2017* (2018).

[2] 2023. LabsLand: the global remote STEM labs platform. [Online] https://labsland.com/en (Aug. 2023).

[3] Jason Blocklove, Siddharth Garg, Ramesh Karri, and Hammond Pearce. 2023. Chip-Chat: Challenges and Opportunities in Conversational Hardware Design. *preprint arXiv:2305.13243* (2023).

[4] Vassilis Fotopoulos et al. 2015. Remote FPGA Laboratory Course Development Based on an Open Multimodal Laboratory Facility. In *Proc. of the PCI Conf.*

[5] Neel Gala and Marc Karasek. 2023. RISC-V Architecture Test SIG. [Online] https://github.com/riscv-non-isa/riscv-arch-test (Sept. 2023).

[6] Sumit Gulwani, Ivan Radiček, and Florian Zuleger. 2018. Automated Clustering and Program Repair for Introductory Programming Assignments. In *Proc. of PLDI Conf.*

[7] Nina Hollender, Cristian Hofmann, Michael Deneke, and Bernhard Schmitz. 2010. Integrating cognitive load theory and concepts of human–computer interaction. *Computers in Human Behavior.*

[8] RISC-V International. 2023. riscv-tests. [Online] https://github.com/riscv-software-src/riscv-tests (Sept. 2023).

[9] Meinhard Kissich, Klaus Weinbauer, and Marcel Baunach. 2023. ATTEST: Automated and Thorough Testing of Embedded Software in Teaching *(ECSEE '23).*

[10] Wade D. Peterson. 2010. *Wishbone B4, WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores.* Silicore Corp.

[11] Tobias Scheipel, Leandro Batista Ribeiro, Tim Sagaster, and Marcel Baunach. 2022. SmartOS: An OS Architecture for Sustainable Embedded Systems. In *Tagungsband des FG-BS Frühjahrstreffens.*

[12] InCore Semiconductors. 2023. RISCOF – The RISC-V Compatibility Framework . [Online] https://riscof.readthedocs.io (Sept. 2023).

[13] Haoye Tian and other. 2023. Is ChatGPT the Ultimate Programming Assistant–How far is it? *preprint arXiv:2304.11938* (2023).

[14] Lionel Sujay Vailshery. 2022. Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030, by use case. [Online] https://www.statista.com/statistics/1194701/iot-connected-devices-use-case (Mar. 2023).

[15] Ignacio Del Villar, Luis Rodriguez-Gil, and Pablo Orduña. 2022. Learning CAN bus communication with a remote laboratory. In *EDUCON Conf.*

[16] Andrew Waterman and Krste Asanovic. 2020. *The RISC-V Instruction Set Manual, Volume I: Unprivileged-Level ISA, Version 20191214-draft.* Technical Report. SiFive Inc., UC Berkeley.